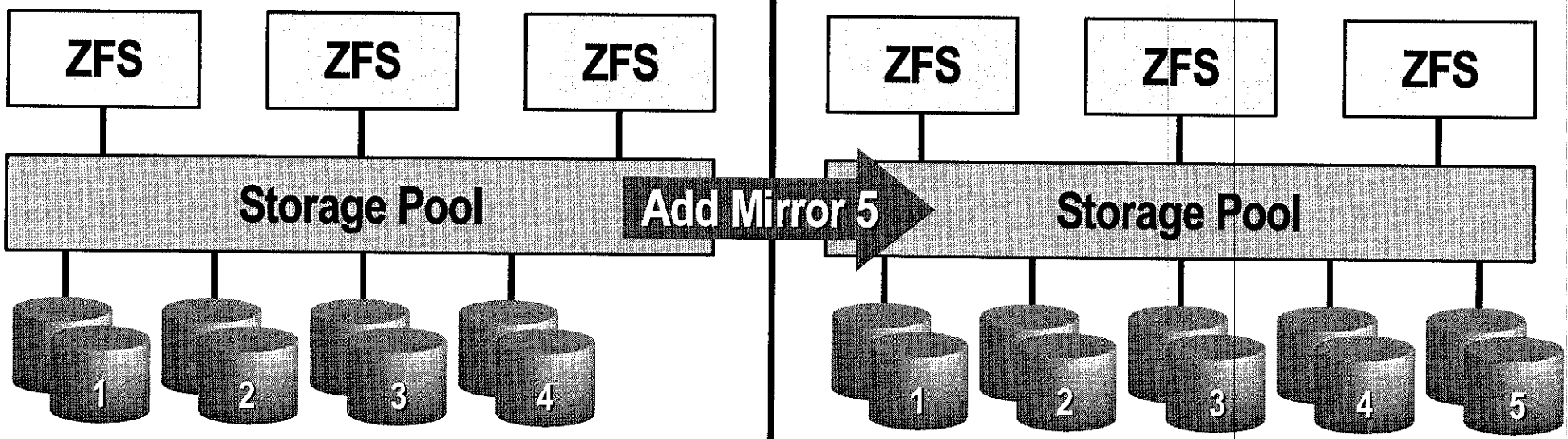
Dynamic Striping

- **Automatically distributes load across all devices**

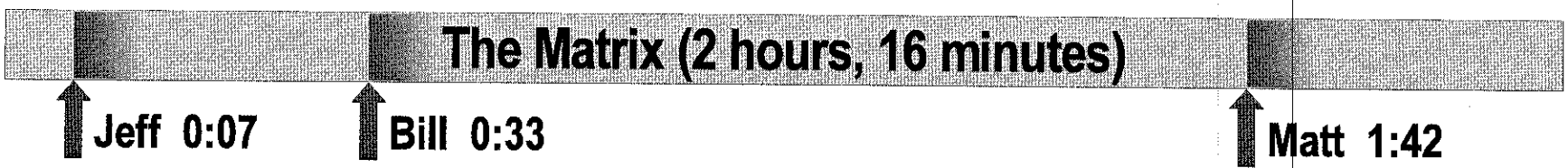
- Writes: striped across all four mirrors
- Reads: wherever the data was written
- Block allocation policy considers:
  - Capacity
  - Performance (latency, BW)
  - Health (degraded mirrors)

- Writes: striped across all five mirrors
- Reads: wherever the data was written
- No need to migrate existing data
  - Old data striped across 1-4
  - New data striped across 1-5
  - COW gently reallocates old data

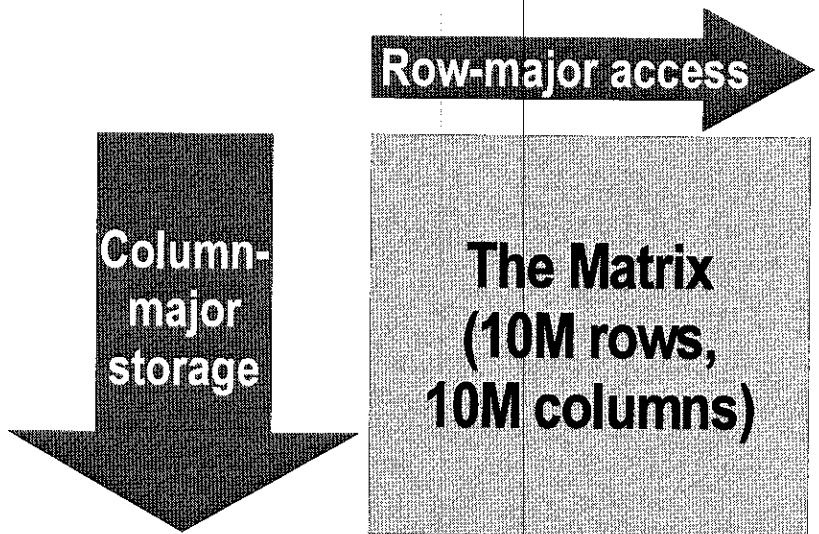


Intelligent Prefetch

- Multiple independent prefetch streams
  - Crucial for any streaming service provider



- Automatic length and stride detection
  - Great for HPC applications
  - ZFS understands the matrix multiply problem
    - Detects any linear access pattern
    - Forward or backward



ZFS Administration

- Pooled storage – no more volumes!
  - All storage is shared – no wasted space, no wasted bandwidth
- Filesystems become administrative control points
  - Hierarchical, with inherited properties
    - Per-dataset policy: snapshots, compression, backups, privileges, etc.
    - Who's using all the space? du(1) takes forever, but df(1M) is instant
  - Control compression, checksums, quotas, reservations, and more
  - Delegate administrative privileges to ordinary users
  - Policy follows the data (mounts, shares, properties, etc.)
  - Manage logically related filesystems as a group
    - Inheritance makes large-scale administration a snap
- Online everything

Creating Pools and Filesystems

- Create a mirrored pool named "tank"

```
# zpool create tank mirror c0t0d0 c1t0d0
```

- Create home directory filesystem, mounted at /export/home

```
# zfs create tank/home
# zfs set mountpoint=/export/home tank/home
```

- Create home directories for several users

Note: automatically mounted at /export/home/{ahrens,bonwick,billm} thanks to inheritance

```
# zfs create tank/home/ahrens
# zfs create tank/home/bonwick
# zfs create tank/home/billm
```

- Add more space to the pool

```
# zpool add tank mirror c2t0d0 c3t0d0
```

Setting Properties

- Automatically NFS-export all home directories

```
# zfs set sharenfs=rw tank/home
```

- Turn on compression for everything in the pool

```
# zfs set compression=on tank
```

- Limit Eric to a quota of 10g

```
# zfs set quota=10g tank/home/eschrock
```

- Guarantee Tabriz a reservation of 20g

```
# zfs set reservation=20g tank/home/tabriz
```

ZFS Snapshots

- Read-only point-in-time copy of a filesystem
  - Instantaneous creation, unlimited number
  - No additional space used – blocks copied only when they change
  - Accessible through .zfs/snapshot in root of each filesystem
    - Allows users to recover files without sysadmin intervention
- Take a snapshot of Mark's home directory

```
# zfs snapshot tank/home/marks@tuesday
```

- Roll back to a previous snapshot

```
# zfs rollback tank/home/perrin@monday
```

- Take a look at Wednesday's version of foo.c

```
$ cat ~maybee/.zfs/snapshot/wednesday/foo.c
```

ZFS Clones

- Writable copy of a snapshot
  - Instantaneous creation, unlimited number
  - Ideal for storing many private copies of mostly-shared data
    - Software installations
    - Workspaces
    - Diskless clients
- Create a clone of your OpenSolaris source code

```
# zfs clone tank/solaris@monday tank/ws/lori/fix
```

ZFS Send / Receive (Backup / Restore)

- Powered by snapshots
  - Full backup: any snapshot
  - Incremental backup: any snapshot delta
    - Very fast – cost proportional to data changed
- So efficient it can drive remote replication
- Generate a full backup

```
# zfs send tank/fs@A >/backup/A
```

- Generate an incremental backup

```
# zfs send -i tank/fs@A tank/fs@B >/backup/B-A
```

- Remote replication: send incremental once per minute

```
# zfs send -i tank/fs@11:31 tank/fs@11:32 |
  ssh host zfs receive -d /tank/fs
```

ZFS Data Migration

- Host-neutral on-disk format
  - Change server from x86 to SPARC, it just works
  - Adaptive endianness: neither platform pays a tax
    - Writes always use native endianness, set bit in block pointer
    - Reads byteswap only if host endianness != block endianness
- ZFS takes care of everything
  - Forget about device paths, config files, /etc/vfstab, etc.
  - ZFS will share/unshare, mount/unmount, etc. as necessary
- Export pool from the old server

```
old# zpool export tank
```

- Physically move disks and import pool to the new server

```
new# zpool import tank
```

ZFS Data Security

- NFSv4/NT-style ACLs
  - Allow/deny with inheritance
- Authentication via cryptographic checksums
  - User-selectable 256-bit checksum algorithms, including SHA-256
  - Data can't be forged – checksums detect it
  - Uberblock checksum provides digital signature for entire pool
- Encryption (coming soon)
  - Protects against spying, SAN snooping, physical device theft
- Secure deletion (coming soon)
  - Thoroughly erases freed blocks

ZFS Root (snv_62)

- Brings all the ZFS goodness to /
  - Checksums, compression, replication, snapshots and clones
  - Boot from any dataset
- Patching becomes safe
  - Take snapshot, apply patch... rollback if you don't like it
- Live upgrade becomes fast
  - Create clone (instant), upgrade, boot from clone
  - No “extra partition”
- Based on new Solaris boot architecture
  - ZFS can easily create multiple boot environments
  - GRUB can easily manage them

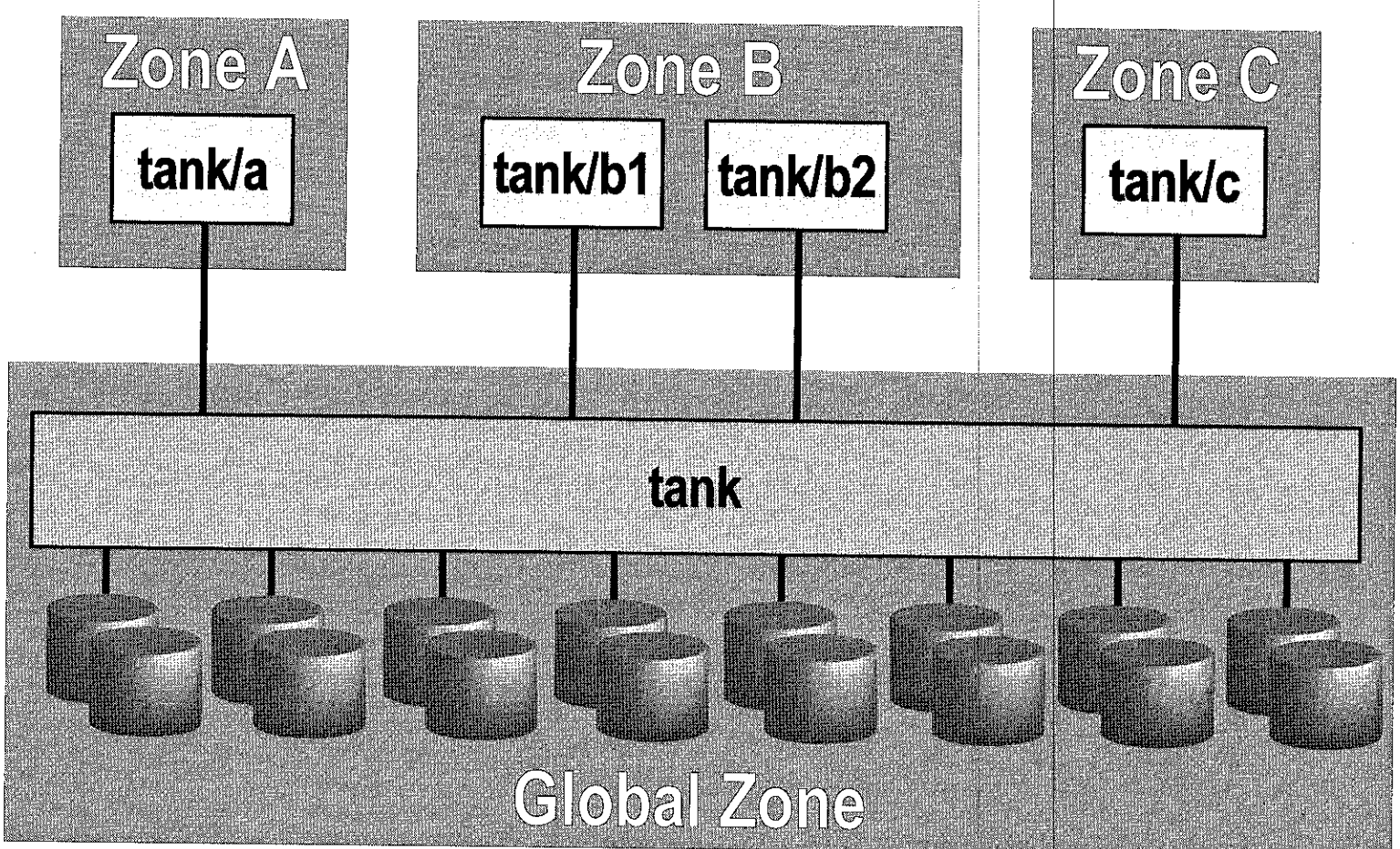
ZFS and Zones (Virtualization)

Strong security model

Local zones cannot even see physical devices

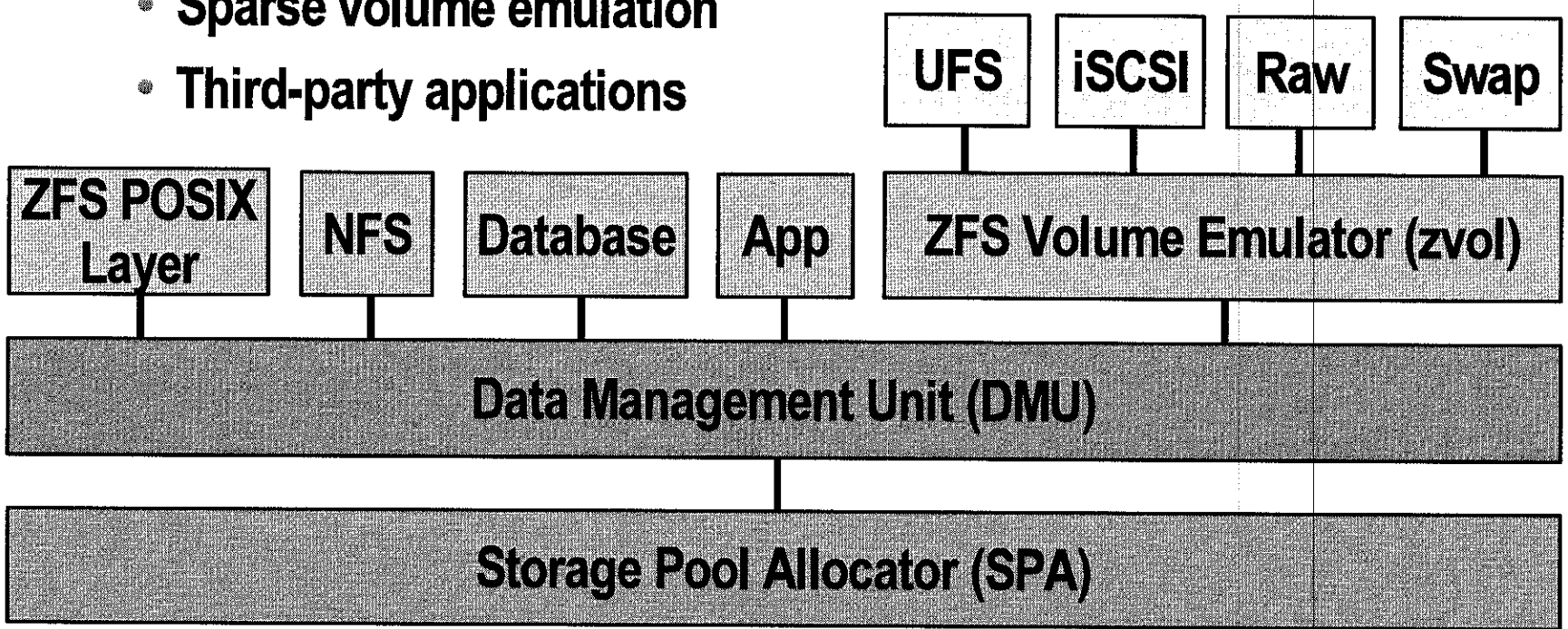
Dataset: Logical resource in local zone

Pool: Physical resource in global zone



Object-Based Storage

- DMU is a general-purpose transactional object store
  - Filesystems
  - Databases
  - Swap space
  - Sparse volume emulation
  - Third-party applications



ZFS Test Methodology

- **A product is only as good as its test suite**
  - **ZFS was designed to run in either user or kernel context**
  - **Nightly "ztest" program does all of the following in parallel:**
    - Read, write, create, and delete files and directories
    - Create and destroy entire filesystems and storage pools
    - Turn compression on and off (while filesystem is active)
    - Change checksum algorithm (while filesystem is active)
    - Add and remove devices (while pool is active)
    - Change I/O caching and scheduling policies (while pool is active)
    - Scribble random garbage on one side of live mirror to test self-healing data
    - Force violent crashes to simulate power loss, then verify pool integrity
  - **Probably more abuse in 20 seconds than you'd see in a lifetime**
  - **ZFS has been subjected to** over a million forced, violent crashes without losing data integrity or leaking a single block

ZFS Summary

End the Suffering • Free Your Mind

- **Simple**
  - Concisely expresses the user's intent
- **Powerful**
  - Pooled storage, snapshots, clones, compression, scrubbing, RAID-Z
- **Safe**
  - Detects and corrects silent data corruption
- **Fast**
  - Dynamic striping, intelligent prefetch, pipelined I/O
- **Open**
  - http://www.opensolaris.org/os/community/zfs
- **Free**

How to Contribute

- http://opensolaris.org/os/community/zfs/
  - Source
  - Admin guide
  - On-disk format guide
  - Porting info
- zfs-discuss@opensolaris.org
  - Feedback
  - Features you want
  - Code contributions
  - Questions

ZFS

THE LAST WORD IN FILE SYSTEMS

eric kustarz

www.opensolaris.org/os/community/zfs